Fly Sense

Individual Lab Report 1

Harikrishnan Suresh

13th October 2017

Sensors and Motor Control Lab

My primary contribution to the sensors and motor control lab assignment was developing the code for IR proximity sensor and servo motor. I also collaborated with the rest of the team in hardware integration.

1. IR proximity sensor

The IR proximity sensor used for this assignment was a Sharp GP2Y0A21YK with a working range of 10-80cm, and is shown in Figure 1.
Working principle: The sensor emits infrared radiations, which return after striking some obstacle and is collected by the receiver. This causes the sensor to give out a voltage, which can be used to infer the distance from the obstacle.

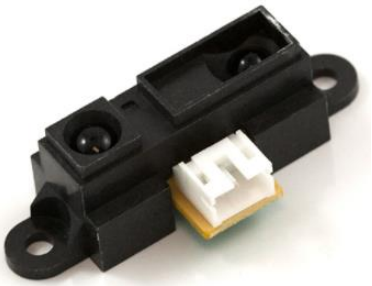

Figure 1: Sharp GP2Y0A21YK

Firstly, the sensor was interfaced with Arduino and the analog readings were noted. Due to large number of erroneous readings coming from the sensor, a median filter was implemented. This involved taking 25 sensor readings, performing bubble sort on them to determine the median value and considering only the median as a valid reading. The next step was to convert the analog readings (0 to 1023) to voltage (0 to 5V). Using this, the calibration of the sensor was done. For the calibration, a set of distances were considered and the corresponding voltage values were noted. Based on the above calibration experiment, a transfer function plot was created in MATLAB and is given in Figure 2. This is in close resemblance with the transfer function plot provided in the data sheet (Figure 3), though it is evident that the working range for the sensor is now 13.24cm to 74.75cm.

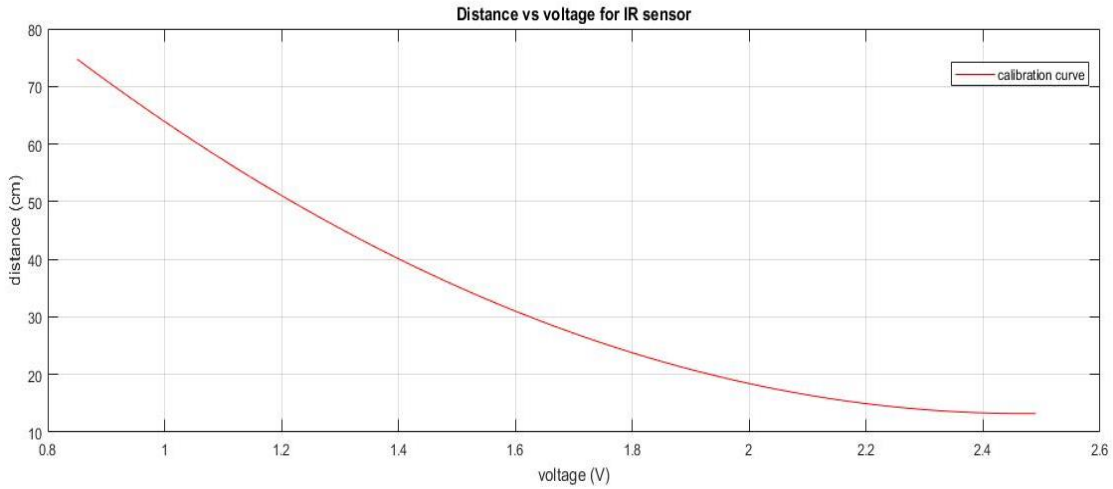

Figure 2: Transfer function plot from experiment

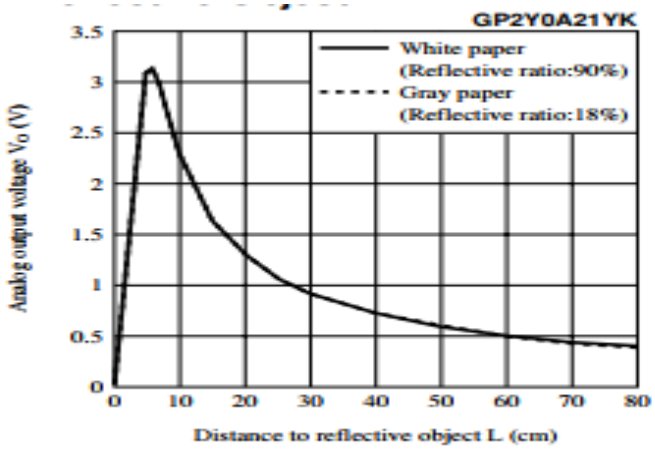

Figure 3: Transfer function plot from datasheet

For this working range of the sensor, the relation between voltage and the distance in cm is given by the following equation,

$$IRdistance = 23.4\ (IRvolt)^2 - 115.7\ (IRvolt) + 156.2$$

The same formula was added to the Arduino code to estimate the distance from the obstacle.

For motor control, a linear relationship between voltage and current was essential. So, using the transfer function plot, it was decided to use only distances between 40 and 70cm as input for the servo motor.

2. Servo motor

The servo motor used for the assignment was an HS-311 shown in Figure 4. The servo has a range of 0-180°
Working principle: The angular position of the servo motors is controlled using pulse width modulation. Servos expect a pulse every 20ms. Length of the pulse width determines the target angular position. A pulse width of 1ms keeps it at 0deg and 2ms keeps it at 180deg. Varying the pulse width between 1ms and 2ms can drive the servo to the angles in between.

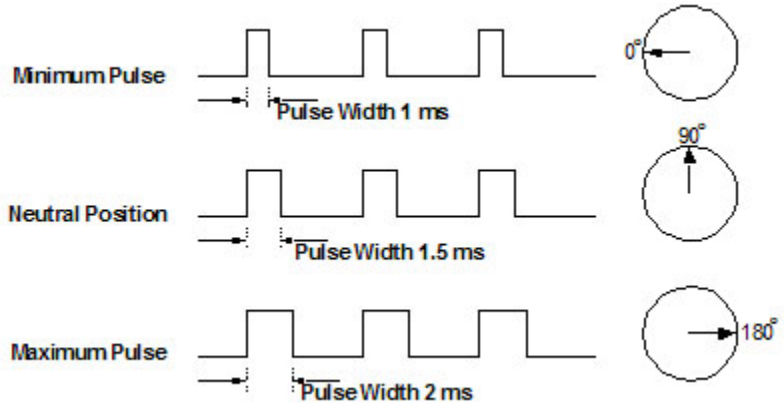

Figure 4: Servo motor working principle

First step was to interface the servo with the Arduino circuit that already had the IR sensor circuit. As the servo is not attached to any load, it can be directly powered by the Arduino 5V. The main purpose was to control the angular position of servo based on the distance shown by the IR sensor. As mentioned earlier, only a range of 40-70 cm was considered as valid inputs for the servo position control. To simplify the programming, the Servo library in Arduino was imported to perform all the servo commands. It performs all the steps mentioned in the working principle, but allows us to program the servo with simple commands. The target position for the servo was determined using a map function in Arduino, that converts distance (40 cm - 70 cm) to angles (0°-180°).

The portion of the code written by me has been included in the appendix A1, along with the main code in appendix A2.

3. Final integration

I collaborated with the rest of the team in the overall integration of both the hardware and software, and assisted in the final debugging process of the main code.

The final hardware set up for the lab is shown in Figure 5.

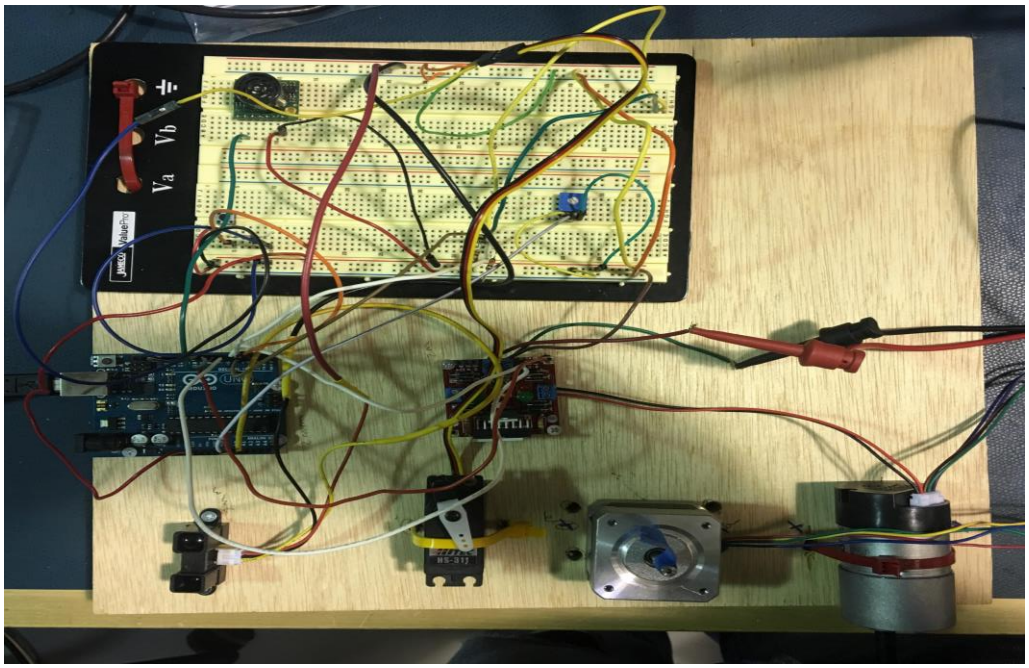

Figure 5: Final setup

4. Challenges faced

1. As seen in the transfer function plot, the IR sensor showed erroneous values below 13.24cm and after 74.75cm. I tried to apply some error corrections to make it work for all close ranges, but nature of the response hindered all attempts. So, through repeated trial and error methods, the working range was chosen.

2. Since the angular positions were mapped directly with the distance values, the servo responded to the commands even out of its working range. This was because of encountering a distance value in the working range (say 45cm) even though the actual distance was below 13.25 cm.

3. The medial filter implemented initially showed some errors due to the fast loop rate of the main code. Since Shivang had implemented a median filter for the ultrasonic sensor that takes only 3 inputs on the go and performs filtering (using a library medianfilter.h), the same was applied for the IR sensor as well.

5. Team roles

The tasks were distributed amongst the team members in the following manner:

Shivang: Responsible for ultrasonic sensor, potentiometer and DC motor control. Also, the primary owner in integration of all the codes together and communication with GUI.

Joao: Collaborated with Nihar for the GUI and communication network. Assisted in final tuning of PID and setting up the hardware.

Nick: Responsible for Force sensor and stepper motor control. Also, the primary owner in hardware integration.

Nihar: Responsible for the GUI and setting communication protocols between GUI and main code.

MRSD project progress

My primary role in the project is in sensing and hardware, along with Shivang. The most important component in the sensing system is the LIDAR, which will be used to detect and track obstacles in real time. For the fall validation experiment and the scale model testing (in quadcopter), we are planning to use Velodyne Puck VLP-16.

We were successful in interfacing the LIDAR with ROS and observing the raw point cloud data in Rviz. Setting up the sensor had some complications, which eventually turned out to be clashes in IP address. We have prepared a document on the various steps to be followed to get the sensor up and running in ROS, which could later be used by other teams working with the same sensor. The point cloud data produced by the LIDAR when placed inside the MRSD lab is shown in Figure 6.

We have also been going through some existing literature on ways to approach the obstacle detection problem, and narrowing down the possible options. One approach would be to focus only on real time tracking without performing any 3D mapping, while the other approach would require an accurate 3D map and segmentation.

We also tried out two ROS packages that take the 3D point cloud data and process it to find the obstacles in the environment. These were just trials on investigating whether the existing algorithms will be useful for our purpose or not.

velodyne_height_map - This ROS package uses a height map algorithm to take the point cloud data, detect the obstacles and clear spaces around. The results obtained after running this package are shown in Figure 7. The region with obstacles and clear spaces are replaced by a set of points in 3D, and can be distinguished. But it does not give a clear and direct segmentation of the environment. For pilots, such an obstacle data will prove to be ineffective unless we run another algorithm over this.

octomap - This ROS package implements a 3D occupancy grid mapping using the concept of octrees. To get this working, the launch file of octree was modified - frame of the map was made same as the frame of LIDAR, and the topic to which the map subscribed to was renamed to the topic published by the LIDAR. The resulting map is shown in Figure 8. The octomap produced a 2D occupancy grid map instead of 3D map, which calls for further modifications to the node.

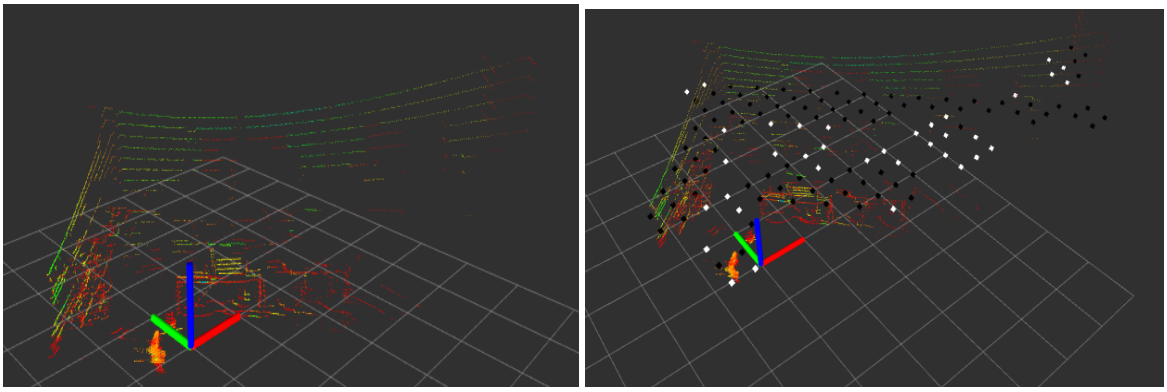

Figure 6: Velodyne point cloud (left), Velodyne_height_map (right)

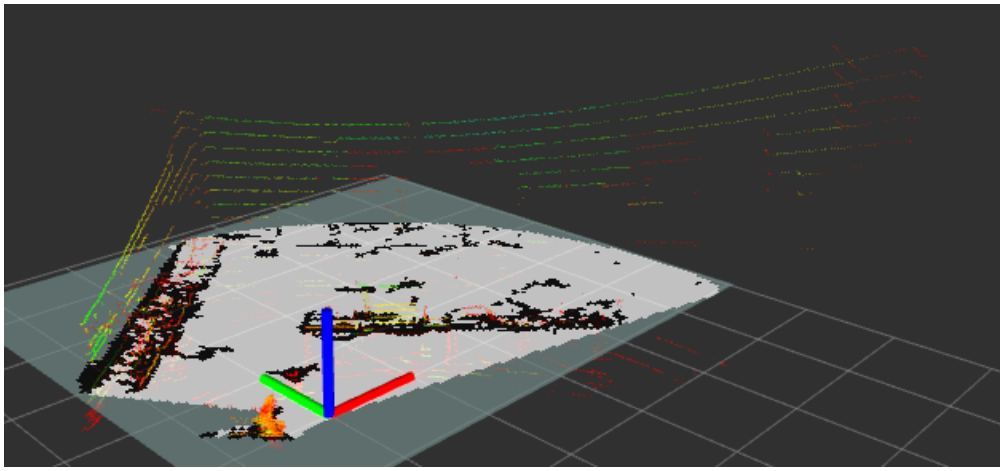

Figure 7: octomap

Further steps

In the coming weeks, we plan to research more into the obstacle detection and tracking algorithms and narrow down to the best algorithm for our case. Upon choosing the algorithm, we will dedicate complete focus to its implementation in ROS to demonstrate it during the fall validation experiment.
We will also be processing the LIDAR data supplied by Near Earth Autonomy collected during their field tests with helicopter.
Shivang and I also plan to acquire the remaining sensors for state estimation and develop the code for the same. We will also design mounts for the LIDAR and other sensors to place on our testing system for fall validation experiment. We will also be working on the power distribution board for the same.

Quiz

1. Reading a datasheet. Refer to the ADXL335 accelerometer datasheet (https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf) to answer the below questions.

- What is the sensor's range?

Minimum : +-3g, Typical: +-3.6g

- What is the sensor's dynamic range?

20log(0.707*3.6/150 ug)

- What is the purpose of the capacitor C_{DC} on the LHS of the functional block diagram on p. 1? How does it achieve this?

The capacitor C_{DC} is used to remove incoming ripple noise from the power supply thereby preventing errors in measurement of acceleration.

- Write an equation for the sensor's transfer function.

The transfer function equation is given by V = 0.3(V/g) * g + 1.5

- What is the largest expected nonlinearity error in g?

Largest expected nonlinearity = +-0.3% of Full scale = +-0.018

- How much noise do you expect in the X- and Y-axis sensor signals when the sensor is excited at 25 Hz?

For x_{OUT}, $y_{OUT,}$ the typical noise level is 150 micro g / sqrt (Hz).

Thus, for 25 Hz the expected noise level should be = 150 * 10^-6 * Sqrt(25) = 0.00015 * 5 = .000750 g

- How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?

The noise level for 0 Hz is not provided in the datasheet. However, static tests can be carried out to determine the sensor's noise. To do this, we can keep the sensor fixed to a surface and noting the readings in each axis. The error estimate can be obtained using standard techniques like root mean square.

2. Signal conditioning

- Filtering
  - What problem(s) might you have in applying a moving average?
    - Moving average is useful for high frequency, low amplitude noise. But the window size must be large, and this causes a delay in the system and is not application for real time systems.

- Moving average filter can cause a shift in the spikes
- Moving average can also cause loss of valuable information (a relevant peak)

- What problem(s) might you have in applying a median filter?
  - Median filters are very useful in eliminating random occurrences of outliers, but multiple occurrences will cause it to fail and give unreasonable data
  - Median filter requires the data in the window to be sorted, and leads to extra computation
  - The effectiveness of median filter depends heavily on the size of the window.

- Opamps
  - In the following questions, you want to calibrate a linear sensor using the circuit in Fig. 1 so that its output range is 0 to 5V. Identify which of V1 and V2 will be the input voltage and which the reference voltage, the value of the reference voltage, and the value of Rf/Ri in each case. If the calibration can't be done with this circuit, explain why.

    Writing the basic opamp voltage equation,
    Vout = V2 + (Rf/Ri)*(V2-V1)

    Considering all the cases:
    - If V2 is Vref, V1 is input,
      - V2 is positive, Vout is positive (not 0) when V1 is negative
      - V2 is negative, Vout is negative when V1 is positive
    - If V1 is Vref, V2 is input,
      - V1 is positive, Vout is negative (not 0) when V2 is negative
      - V1 is negative satisfies, both the negative and positive inputs. So, V1 must be negative.

    - Your uncalibrated sensor has a range of -1.5 to 1.0V.

    → To achieve the output range (0 to 5v) mapped to input range (-1.5 to 1)
    V1 is Vref and V1=-3
    Rf/Ri=1

    - Your uncalibrated sensor has a range of -2.5 to 2.5V.
      Vout = V2 + (Rf/Ri)*(V2 + V1)

      5 = -2.5 + (Rf/Ri)*(-2.5 +V1)
      0 = 2.5 + (Rf/Ri)*(2.5 +V1)

      Adding the two equations,
      We get, (Ri/Rf)*V1=2.5

      Since resistance can't be negative, V1 has to be positive. If V1 is positive we don't have a solution which maps to output (0 to 5V), as described above.

3. Control

- If you want to control a DC motor to go to a desired position, describe how to form a digital input for each of the PID (Proportional, Integral, Derivative) terms.

  1. Calculate error using the encoder values and applying the following formula
  Error = Target position – Current position

  2. Integrate the error, but only till the control saturates
  3. Take differential of the error value.
  4. Control input = Pgain * error + Igain* Integrator + Dgain * Error rate (differential of error)

- If the system you want to control is sluggish, which PID term(s) will you use and why?

If the system is slow, the Proportional gain must be increased to speed up the response.

- After applying the control in the previous question, if the system still has significant steady-state error, which PID term(s) will you use and why?

Adding an integral gain reduces the steady state error

- After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?

Adding a derivative gain causes damping and can help reduce oscillations and overshoot.

Appendix A1

```
#include <Servo.h>
Servo myservo;

const int ir=A0;
const int servo=9;
float IRinput[25];
float IRdistance;
float IRvolt,IRvolt1;
float tmppos;
int pos;

void setup() {
myservo.attach(9);
pinMode(ir, INPUT);
Serial.begin(9600);
}

void sort(float a[])
{
   for(int i=0; i<25; i++)
   {
      bool flag = true;
      for(int j=0; j<(25-(i+1)); j++)
      {
         if(a[j] > a[j+1])
         {
            int t = a[j];
            a[j] = a[j+1];
            a[j+1] = t;
            flag = false;
         }
      }
      if (flag) break;
   }

}

void loop()
{
for (int i=0; i<25; i++)
   {
      IRinput[i] = analogRead(ir);
   }
```

```
sort(IRinput);

IRvolt = map((IRinput[13]+IRinput[12])/2.0,0,1023,0,5000);
IRvolt1 = IRvolt/1000.0;
if(IRvolt1 >=0.85 && IRvolt1<=2.5)
{
  IRdistance = 23.4 * IRvolt1 * IRvolt1 -115.7*IRvolt1 + 156.2; // from transfer function
    tmppos = map(IRdistance,40.0,70.0,0.0,180.0);
    myservo.write(tmppos);

    delay(50);
}
else
{
  IRdistance=-100;
}

delay(1000);

}
```